PASOS DABD 2 PARCIAL

Creamos clase ciudad para el nuevo combo box

-ng g class models/ciudad

Ciudad.ts

```
export class Ciudad {
    id: string;
    nombre: string;
}
```

Persona.ts

```
export class Persona {
    id: string;
    nombre: string;
    apellido: string;
    edad: number;
    paisId: string;
    pais: Pais;
    ciudadId:string; //agregamos estos
    ciudad:Ciudad;//Este solo si se requiere
}
```

Generamos el servicio de Ciudad

-ng g s services/ciudad

Copiamos el servicio de país

ciudad.service.ts

```
export class CiudadService {
  private API_URL: string = 'https://636c46a7ad62451f9fc6d36f.mockapi.io/pais/';
  constructor(private http: HttpClient) {}
  obtenerCiudadesPorPais(idPais:number): Observable<Ciudad[]> { //se carga por medio
del parametro enviado desde el alta (parametro id pais)
    return this.http.get<Ciudad[]>(this.API_URL+idPais+'/ciudad');
  }
}
```

app.module.ts

```
 imports: [
    BrowserModule,
    AppRoutingModule,
    //FormsModule,//ELIMINAMOS ESTE
    ReactiveFormsModule,//Agregamos este
    HttpClientModule
  ]
```

MODIFICAMOS EL ALTA CON REACTIVE FORMS

Borramos el viewChild, cambiamos el this.persona por formulario.value y en el cargar agregamos el patch.value

Agregar arreglo de ciudad su servicio y el cargarComboCiudad 😉

BORRAR LOS IMPORT QUE NO SE USEN!

persona-alta.component.ts

```
  formulario: FormGroup;
  ciudades:Ciudad[];//Agregamos esto en la clase Persona
```

-En dentro de () del constructor agregamos:

```
  constructor(
    private FormBuilder:FormBuilder,//reactive form
    private activatedRoute: ActivatedRoute, //para el metodo cargar
    private ciudadService: CiudadService,//para obtener las ciudades
```

-En el OnInit agregamos el cargar Cbociudad

```
  cargarCboCiudadPorPais() : void{
    this.formulario.controls['paisId'].valueChanges.subscribe(x=>{
      this.subscription.add(
        this.ciudadService.obtenerCiudadesPorPais(x).subscribe({
          next : (r : Ciudad[]) =>{
            this.ciudades=r;
          },
          error :(e) =>{
            console.error(e);
          }
        })
      )
    })
  }
```

-y el cargar formulario para si es que se pide editar:

```
cargar () : void{ //CARGAMOS POR ID EN EL FORMULARIO
    this.activatedRoute.params.subscribe(
      e=>{
        let id = e['id'];
        if(id){
          this.isEdit=true;
          this.personaService.getPersonaById(id).subscribe(
            es => {
              this.persona= es;
              this.formulario.patchValue(this.persona)
            }
        )}else{
          this.isEdit=false;
        }
      }
    )
  }
```

persona.service.ts

```
getPersonaById (id:number) : Observable <Persona>{
    return this.http.get<Persona>(this.URL+id)
  }
```

En el alta cargamos el formbuilder en el oninit:

```
  ngOnInit(): void {
    this.formulario = this.formBuilder.group({
      nombre : [null,
      [Validators.required],  //validacion sincronica que es requerido el
campo
      [PersonaValidador.nombreValidador(this.personaService)]//validacion
asincronica depende de lo que pide el profe
      ],
      apellido : [null,
        [Validators.required],
        [PersonaValidador.apellidoValidador(this.personaService)]
        ],
      edad : [,Validators.required],
      precio : [,Validators.required],
      fecha:[,Validators.required],
      paisId : [,Validators.required],
      ciudadId : [,Validators.required],
    })
```

Modificamos el HTML del alta

Borramos ngForm y ponemos:

```
<form [formGroup]="formulario">
```

Borramos los required de los input borramos el campo name y lo cambiamos por formControlName y borramos todo lo que contenga Ng, agregamos el cboCiudad

```
    <div class="mb-3">
        <label for="selectCiudad" class="form-label">Ciudad</label>
        <select id="selectCiudad" class="form-select" formControlName="ciudadId" >
          <option *ngFor="let c of ciudades" [value]="c.id">{{ c.nombre }}</option>
        </select>
      </div>
```

VALIDACION async

En el servicio de persona agregamos los métodos para validar

persona.service.ts

```
//Validaciones
  nombreExiste(valor : string) : Observable<boolean>{
    return this.http.get<Persona[]>(this.API_URL).pipe(
      map(x => x.some((persona)=> persona.nombre == valor))
    )
  }
  //no se pueden mas de 3 con el mismo apellido
  apellidoExisten3 (valor :string ): Observable<boolean> {
    return this.http.get<Persona[]>(this.API_URL).pipe(
      map((arregloPersona: Persona[])=> (arregloPersona.filter((persona: Persona)=>
persona.apellido== valor).length > 3 ))
    )
  }
```

Luego creamos una clase validator ng g class validators/persona-validador y agregamos los siguientes metodos con sus respectivos imports.

persona-validador.ts

```
static nombreValidador(servicioPersona: PersonaService): AsyncValidatorFn {
        return (control: AbstractControl): Observable<ValidationErrors | null> => {
          return servicioPersona
            .nombreExiste(control.value)
            .pipe(
              map((result: boolean) =>
                result ? { nameAlReadyExists: true } : null
              )
            );
        };
      }
      static apellidoValidador(servicioPersona: PersonaService): AsyncValidatorFn {
        return (control: AbstractControl): Observable<ValidationErrors | null> => {
          return servicioPersona
            .apellidoExisten3(control.value)
            .pipe(
              map((result: boolean) =>
                result ? { lastNameAlReadyExists: true } : null
              )
            );
        };
      }
```

Recordar agregar en el formBuilder.group del alta y en el formulario html agregar lo siguiente que es la validación Ej:

APELLIDO INPUT con validación sincrónica y asincronica

```
    <div class="mb-3">
       <label for="inputApellisdo" class="form-label">Apellido</label>
       <input  type="text" class="form-control" id="inputApellido" formControlName="apellido"
       [class.is-invalid]="controlApellido.touched&&controlApellido.hasError('required')"
       [class.is-
invalid]="controlApellido.touched&&controlApellido.hasError('lastNameAlReadyExists')">
    </div>
    <span *ngIf="controlApellido.touched&&controlApellido.hasError('required')">
       Apellido es requerido
     </span>
     <span *ngIf="controlApellido.touched&&controlApellido.hasError('lastNameAlReadyExists')">
      no pueden existir mas de 3 apellidos
     </span>
```

Luego en el ts de alta agregamos los get requerido para cada async

```
get controlNombre (): FormControl{
    return this.formulario.controls['nombre'] as FormControl;
  }

  get controlApellido (): FormControl{
    return this.formulario.controls['apellido'] as FormControl;
  }
```

GENERACION DE PIPE ng g p pipes/esMayor

```
export class EsMayorPipe implements PipeTransform {
  private readonly limite_por_defecto=18;
  transform(value: number, ...args: any[]): any {
    const limiteHTML = args[0]? args[0] : this.limite_por_defecto;
    const resultado = value >= limiteHTML ? "SI": "NO";
    return resultado;
  }
}
```

En el listado agregamos la columna y la siguiente fila mandando por parámetros un valor al pipe:

persona.edad=value esMayor(llamamos al pipe) 18 es un valor agregado al args[0].

```
            <td>{{persona.edad | esMayor: 18 }}</td>
```

Otros ejemplos de pipes:

```
export class PrecioDescuntoMayorPipe implements PipeTransform {
  private  procentaje_por_defecto=0;
  transform(precioDesc: number, ...args: any[]):number{
    if (args [0]>18) {
      this.procentaje_por_defecto=0.1;
    } else {
      this.procentaje_por_defecto=0;
```

```
    }
    const porcentaje = this.procentaje_por_defecto;
    const resultado =  ( precioDesc-(porcentaje*precioDesc));
    return  resultado;
```

```
export class PrecioMasIvaPipe implements PipeTransform {
  private readonly procentaje_por_defecto=21;
  transform(precioPersona: number, ...args: any[]):number{
    const porcentaje = args [0] ? args [0] : this.procentaje_por_defecto;
    const resultado =  precioPersona * ((porcentaje / 100) +1);
    return resultado;
  }
}
```